Neural Processing Letters (2021) 53:1147–1160
https://doi.org/10.1007/s11063-021-10434-9

KTBoost: Combined Kernel and Tree Boosting

Fabio Sigrist[1]

Accepted: 23 January 2021 / Published online: 11 February 2021

Abstract
We introduce a novel boosting algorithm called 'KTBoost' which combines **k**ernel boosting and **t**ree boosting. In each boosting iteration, the algorithm adds either a regression tree or reproducing kernel Hilbert space (RKHS) regression function to the ensemble of base learners. Intuitively, the idea is that discontinuous trees and continuous RKHS regression functions complement each other, and that this combination allows for better learning of functions that have parts with varying degrees of regularity such as discontinuities and smooth parts. We empirically show that KTBoost significantly outperforms both tree and kernel boosting in terms of predictive accuracy in a comparison on a wide array of data sets.

Keywords Gradient and newton boosting · Reproducing kernel Hilbert space (RKHS) regression · Ensemble learning · Supervised learning

1 Introduction

Boosting algorithms [8,15,17,18,28] enjoy large popularity in both applied data science and machine learning research, among other things, due to their high predictive accuracy observed on a wide range of data sets [11]. Boosting additively combines base learners by sequentially minimizing a risk functional. Despite the fact that there is almost no restriction on the type of base learners in the seminal papers of Freund and Schapire [15] and Freund and Schapire [16], very little research has been done on combining different types of base learners. To the best of our knowledge, except for one reference [22], existing boosting algorithms use only one type of functions as base learners. To date, regression trees are the most common choice of base learners, and a lot of effort has been made in recent years to develop tree-based boosting methods that scale to large data [11,26,35,36].

In this article, we relax the assumption of using only one type of base learners by combining regression trees [7] and reproducing kernel Hilbert space (RKHS) regression functions [4,39] as base learners. In short, RKHS regression is a form of non-parametric regression which shows state-of-the-art predictive accuracy for many data sets as it can, for instance, achieve near-optimal test errors [1,2], and kernel classifiers parallel the behaviors of deep networks as noted in Zhang et al. [46]. As there is now growing evidence that base learners do not

✉ Fabio Sigrist
fabio.sigrist@hslu.ch

[1] Lucerne University of Applied Sciences and Arts, Suurstoffi 1, 6343 Rotkreuz, Switzerland

necessarily need to have low complexity [44], continuous, or smooth, RKHS functions have thus the potential to complement discontinuous trees as base learners.

1.1 Summary of Results

We introduce a novel boosting algorithm denoted by 'KTBoost' which combines **k**ernel and **t**ree boosting. In each boosting iteration, the KTBoost algorithm adds either a regression tree or a penalized RKHS regression function, also known as kernel ridge regression [30], to the ensemble. This is done by first learning both a tree and an RKHS function using one step of functional Newton's method or functional gradient descent, and then selecting the base learner whose addition to the ensemble results in the lowest empirical risk. The KTBoost algorithm thus chooses in each iteration a base learner from two fundamentally different function classes. Functions in an RKHS are continuous and, depending on the kernel function, they also have higher regularity. Trees, on the other hand, are discontinuous functions.

Intuitively, the idea is that the different types of base learners complement each other, and that this combination allows for better learning of functions that exhibit parts with varying degrees of regularity. We demonstrate this effect in a simulation study in Sect. 4.1. To briefly illustrate that the combination of trees and RKHS functions as base learners can achieve higher predictive accuracy, we report in Fig. 1 test mean square errors (MSEs) versus the number of boosting iterations for one data set (wine). The solid lines show average test MSEs over ten random splits into training, validation, and test data sets versus the number of boosting iterations. The confidence bands are obtained after point-wise excluding the largest and smallest MSEs. Tuning parameters of all methods are chosen on the validation data sets. See Sect. 4 for more details on the data set and the choice of tuning parameters.[1] The figure illustrates how the combination of tree and kernel boosting (KTBoost) results in a lower test MSE compared to both tree and kernel boosting. In our extensive experiments in Sect. 4.2, we show on a large collection of data sets that the combination of trees and RKHS functions leads to a lower generalization error compared to both only tree and only kernel boosting. Our approach is implemented in the Python package `KTBoost` which is openly available on the Python Package Index (PyPI) repository.[2]

1.2 Related Work

Combining predictions from several models has been successfully applied in many areas of machine learning such as diversity inducing methods [29] or multi-view learning; see e.g. Peng et al. [34] for a recent example of a boosting application. However, the way boosting combines base learners is different from traditional ensembles consisting of several models trained on potentially different data sets since, for instance, boosting reduces both variance and bias. Very little research has been done on combining different types of base learners in a boosting framework, and, to the best of our knowledge, there is no study which investigates the effect on the predictive accuracy when boosting different types of base learners.

The `mboost` R package of Hothorn et al. [22] allows for combining different base learners which include linear functions, one- and two-dimensional smoothing splines, spatial terms,

[1] For better comparison, the shrinkage parameter ν, see Eq. (4), is set to a fix value ($\nu = 0.1$) in this example. In the experiments in Sect. 4.2, the shrinkage parameter is also chosen using cross-validation.

[2] See https://github.com/fabsig/KTBoost for more information.

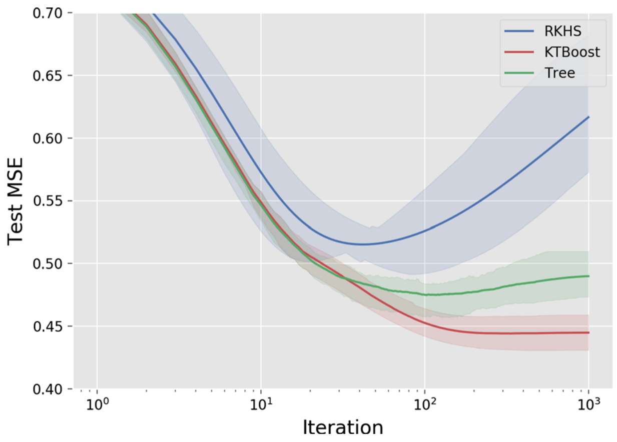

Fig. 1 Test mean square error (MSE) versus the number of boosting iteration for KTBoost in comparison with tree and kernel boosting for one data set (wine)

regression trees, as well as user-defined ones. This approach is different from ours since `mboost` uses a component-wise approach where every base learner typically depends on only a few features, and in each boosting update, the term which minimizes a least squares approximation to the negative gradient of the empirical risk is added to the ensemble. In contrast, in our approach, the tree and the kernel machine depend on all features by default, base learners are learned using Newton's method or gradient descent, and we select the base learner whose addition to the ensemble directly results in the lowest empirical risk.

The idea that machine learning methods should be able learn both smooth as well as non-smooth functions has recently received attention also in other areas of machine learning. For instance, Imaizumi and Fukumizu [24] and Hayakawa and Suzuki [20] argue that one of the reasons for the superior predictive accuracy of deep neural networks, over e.g. kernel methods, is their ability to also learn non-smooth functions.

2 Preliminaries

2.1 Boosting

There exist population as well as sample versions of boosting algorithms. For the sake of brevity, we only consider the latter here. Assume that we have data $\{(x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}, i = 1, \dots, n\}$ from a probability distribution $P_{X,Y}$. The goal of boosting is to find a function $F : \mathbb{R}^p \rightarrow \mathbb{R}$ for predicting y given x, where F is in a function space Ω_S with inner product $\langle \cdot, \cdot \rangle$ given by $\langle F, F \rangle = E_X \left(F(X)^2 \right)$, and the expectation is with respect to the marginal distribution P_X of $P_{X,Y}$. Note that y can be categorical, discrete, continuous, or of mixed type depending on whether the conditional distribution $P_{Y|X}$ is absolutely continuous with respect to the Lebesgue, a counting measure, or a mixture of the two; see, e.g., Sigrist and Hirnschall [42] for an example of the latter. Depending on the data and the goal of the application, the function can also be multivariate. For the sake of notational simplicity, we assume in the following that F is univariate. The extension to the multivariate case $F = (F^k)$, $k = 1, \dots, d$, is straightforward; see, e.g., Sigrist [41].

The goal of boosting is to find a minimizer $F^*(\cdot)$ of the empirical risk functional $R(F)$:

$$F^*(\cdot) = \underset{F(\cdot)\in\Omega_{\mathcal{S}}}{\operatorname{argmin}} R(F) = \underset{F(\cdot)\in\Omega_{\mathcal{S}}}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F(x_i)), \tag{1}$$

where $L(Y, F)$ is an appropriately chosen loss function such as the squared error for regression or the logistic regression loss for binary classification, and $\Omega_{\mathcal{S}} = span(\mathcal{S})$ is the span of a set of base learners $\mathcal{S} = \{f_j : \mathbb{R}^p \to \mathbb{R}\}$. Boosting finds $F^*(\cdot)$ in a sequential way by iteratively adding an update f_m to the current estimate F_{m-1}:

$$F_m(x) = F_{m-1}(x) + f_m(x), \quad f_m \in \mathcal{S}, \quad m = 1, \ldots, M, \tag{2}$$

such that the empirical risk is minimized

$$f_m = \underset{f\in\mathcal{S}}{\operatorname{argmin}} R\left(F_{m-1} + f\right). \tag{3}$$

Since this usually cannot be found explicitly, one uses an approximate minimizer. Depending on whether gradient or Newton boosting is used, the update f_m is either obtained as the least squares approximation to the negative functional gradient or by applying one step of functional Newton's method which corresponds to minimizing a second order Taylor expansion of the risk functional; see Sect. 3 or Sigrist [41] for more information. For increased predictive accuracy [18], an additional shrinkage parameter $\nu > 0$ is usually added to the update equation:

$$F_m(x) = F_{m-1}(x) + \nu f_m(x). \tag{4}$$

2.2 Reproducing Kernel Hilbert Space Regression

Assume that $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a positive definite kernel function. Then there exists a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ with an inner product $\langle\cdot,\cdot\rangle$ such that (i) the function $K(\cdot, x)$ belongs to $\mathcal{H}$ for all $x \in \mathbb{R}^d$ and (ii) $f(x) = \langle f, K(\cdot, x)\rangle$ for all $f \in \mathcal{H}$. Suppose we are interested in finding the minimizer

$$\underset{f\in\mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda\|f\|_{\mathcal{H}}^2, \tag{5}$$

where $\lambda \geq 0$ is a regularization parameter. The representer theorem [40] then states that there is a unique minimizer of the form

$$f(\cdot) = \sum_{j=1}^{n} \alpha_j K(x_j, \cdot)$$

and (5) can be written as

$$\underset{\alpha\in\mathbb{R}^n}{\operatorname{argmin}} \|y - K\alpha\|^2 + \lambda\alpha^T K\alpha,$$

where $y = (y_1, \ldots, y_n)^T$, $K \in \mathbb{R}^{n\times n}$ with $K_{ij} = K(x_i, x_j)$, and $\alpha = (\alpha_1, \ldots, \alpha_n)^T$. Taking derivatives and equaling them to zero, we find the explicit solution as

$$\alpha = (K + \lambda I_n)^{-1} y,$$

where I_n denotes the n-dimensional identity matrix.

There is a close connection between Gaussian process regression and kernel regression. The solution to (5) is the posterior mean conditional on the data of a zero-mean Gaussian process with covariance function K. Further, since

$$f(x) = k(x)^T (K + \lambda I_n)^{-1} y,$$

where

$$k(x) = (K(x_1, x), \ldots, K(x_n, x))^T, \tag{6}$$

kernel regression can also be interpreted as a two-layer neural network.

2.3 Regression Trees

We denote by $\mathcal{T}$ the space which consists of regression trees [7]. Following the notation used in Chen and Guestrin [11], a regression tree is given by

$$f^T(x) = w_{s(x)},$$

where $s : \mathbb{R}^p \to \{1, \ldots, J\}$, $w \in \mathbb{R}^J$, and $J \in \mathbb{N}$ denotes the number of terminal nodes of the tree $f^T(x)$. s determines the structure of the tree, i.e., the partition of the space, and w denotes the leaf values. As in Breiman et al. [7], we assume that the partition of the space made by s is a binary tree where each cell in the partition is a rectangle of the form $R_j = (l_1, u_1] \times \cdots \times (l_p, u_p] \subset \mathbb{R}^p$ with $-\infty \le l_m < u_m \le \infty$ and $s(x) = j$ if $x \in R_j$.

3 Combined Kernel and Tree Boosting

Let $R^2(F_{m-1} + f)$ denote the functional, which is proportional to a second order Taylor approximation of the empirical risk in (1) at the current estimate F_{m-1}:

$$R^2(F_{m-1} + f) = \sum_{i=1}^{n} g_{m,i} f(x_i) + \frac{1}{2} h_{m,i} f(x_i)^2, \tag{7}$$

where $g_{m,i}$ and $h_{m,i}$ are the functional gradient and Hessian of the empirical risk evaluated at the functions $F_{m-1}(x)$ and $I_{\{x=x_i\}}(x)$, where $I_{\{x=x_i\}}(x) = 1$ if $x = x_i$ and 0 otherwise:

$$\begin{aligned} g_{m,i} &= \frac{\partial}{\partial F} L(y_i, F)\Big|_{F=F_{m-1}(x_i)}, \\ h_{m,i} &= \frac{\partial^2}{\partial F^2} L(y_i, F)\Big|_{F=F_{m-1}(x_i)}. \end{aligned} \tag{8}$$

The KTBoost algorithm presented in Algorithm 1 works as follows. In each boosting iteration, a candidate tree $f_m^T(x)$ and RKHS function $f_m^K(x)$ are found as minimizers of the second order Taylor approximation $R^2(F_{m-1} + f)$. This corresponds to applying one step of a functional version of Newton's method. It can be shown that candidate trees $f_m^T(x)$ can be found as weighted least squares minimizers; see, e.g., Chen and Guestrin [11] or Sigrist [41]. Further, the candidate penalized RKHS regression functions $f_m^K(x)$ can be found as shown in Proposition 1 below. The KTBoost algorithm then selects either the tree or the RKHS function such that the addition of the base learner to the ensemble according to Eq. (4) results in the lowest risk. Note that for the RKHS boosting part, the update equation $F_m(x) = F_{m-1}(x) + \nu f_m(x)$ can be replaced by simply updating the coefficients α_m.

Algorithm 1: KTBoost

```
1: Initialize $F_0(x) = \text{argmin}_{c\in\mathbb{R}^d} R(c)$.
2: for $m = 1$ to $M$ do
3:   Compute the gradient $g_{m,i}$ and Hessian $h_{m,i}$ as defined in (8)
4:   Find the candidate regression tree $f_m^T(x)$ and RKHS function $f_m^K(x)$
        $f_m^T(x) = \text{argmin}_{f\in\mathcal{T}} R^2(F_{m-1} + f)$
        $f_m^K(x) = \text{argmin}_{f\in\mathcal{H}} R^2(F_{m-1} + f) + \frac{1}{2}\lambda\|f\|^2_{\mathcal{H}}$
     where the approximate risk $R^2(F_{m-1} + f)$ is defined in (7)
5:   if $R\left(F_{m-1} + \nu f_m^T(x)\right) \leq R\left(F_{m-1} + \nu f_m^K(x)\right)$ then
6:     $f_m(x) = f_m^T(x)$
7:   else
8:     $f_m(x) = f_m^K(x)$
9:   end if
10:  Update $F_m(x) = F_{m-1}(x) + \nu f_m(x)$
11: end for
```

If either the loss function is not twice differentiable in its second argument or the second derivative is zero or constant on a non-null set of the support of X, one can alternatively use gradient boosting. The gradient boosting version of KTBoost is obtained as a special case of the Algorithm 1 by setting $h_{m,i} = 1$. Gradient boosting has the advantage that it is computationally less expensive than Newton boosting since, in contrast to (9), the kernel matrix does not depend on the iteration number m; see Sect. 3.1 for more details.

Proposition 1 *The kernel ridge regression solution $f_m^K(x)$ in the regularized Newton boosting update step is given by $f_m^K(x) = k(x)^T\alpha_m$, where $k(x)$ is defined in (6) and*

$$\alpha_m = D_m\left(D_m K D_m + \lambda I_n\right)^{-1} D_m y_m, \tag{9}$$

where $D_m = diag\left(\sqrt{h_{m,i}}\right)$, $h_{m,i} > 0$, $y_m = (-g_{m,1}/h_{m,1}, \ldots, -g_{m,n}/h_{m,n})^T$, and I_n is the identity matrix of dimension n.

Proof We have

$$\begin{aligned}
&\underset{f\in\mathcal{H}}{\text{argmin}} \sum_{i=1}^n g_{m,i} f(x_i) + \frac{1}{2} h_{m,i} f(x_i)^2 + \frac{1}{2}\lambda\|f\|^2_{\mathcal{H}}\\
&= \underset{f\in\mathcal{H}}{\text{argmin}} \sum_{i=1}^n h_{m,i}\left(-\frac{g_{m,i}}{h_{m,i}} - f(x_i)\right)^2 + \lambda\|f\|^2_{\mathcal{H}}\\
&= \underset{\alpha}{\text{argmin}}\|D_m y_m - D_m K\alpha\|^2 + \lambda\alpha^T K\alpha.
\end{aligned}$$

If we take derivatives with respect to α, equal them to zero, and solve for α, we find that

$$\begin{aligned}
\alpha_m &= \left(K D_m^2 K + \lambda K\right)^{-1} K D_m^2 y_m\\
&= \left(D_m^2 K + \lambda I_n\right)^{-1} D_m^2 y_m\\
&= D_m\left(D_m K D_m + \lambda I_n\right)^{-1} D_m y_m.
\end{aligned}$$

□

3.1 Reducing Computational Costs for Large Data

Concerning the regression trees, finding the splits when growing the trees is the computationally demanding part. There are several approaches in the literature on how this can be done efficiently for large data; see, e.g., Chen and Guestrin [11] or Ke et al. [26]. The computationally expensive part for finding the kernel regression updates is the factorization of the kernel matrix which scales with $O(n^3)$ in time. There are several approaches that allow for computational efficiency in the large data case. Examples of this include low rank approximations based on, e.g., the Nyström method [43] and extensions of it such as divide-and-conquer kernel ridge regression [47,48], early stopping of iterative optimization methods [6,27,38,45], stochastic gradient descent [10,12], random feature approximations [37], and compactly supported kernel functions [5,19] which results in a sparse kernel matrix K which can be efficiently factorized.

Note that if gradient descent is used instead of Newton's method, the RKHS function $f_m^K(x)$ can be found efficiently by observing that, in contrast to (9), the kernel matrix $K+\lambda I_n$ does not depend on the iteration number m, i.e., its inverse or a Cholesky factor of it needs to be calculated only once. Further, the two learners can be learned in parallel.

In our empirical analysis, we use the Nyström method for dealing with large data sets. The Nyström method approximates the kernel $K(\cdot,\cdot)$ by first choosing a set of l so-called Nyström samples $x_1^*,\dots,x_l^*$. Often these are obtained by sampling uniformly from the data. Denoting the kernel matrix that corresponds to these points as K^*, the Nyström method then approximates the kernel $K(\cdot,\cdot)$ as

$$K(x,y) \approx k_l(x)^T {K_{l,l}^*}^{-1} k_l(y),$$

where $k_l(x) = (K(x,x_1^*),\dots,K(x,x_l^*))^T$ and $\left(K_{l,l}^*\right)_{j,k} = K(x_j^*,x_k^*)$, $1 \le j,k \le l$. In particular, the reduced-rank Nyström approximation to the full kernel matrix K is given by

$$K \approx K_{n,l}^* {K_{l,l}^*}^{-1} {K_{n,l}^*}^T,$$

where $\left(K_{n,l}^*\right)_{j,k} = K(x_j,x_k^*)$, $1 \le j \le n$, $1 \le k \le l$.

4 Experimental Results

4.1 Simulation Study

We first conduct a small simulation study to illustrate that the combination of discontinuous trees and continuous kernel machines can indeed better learn functions with both discontinuous and smooth parts. We consider random functions $F : [0,1] \to \mathbb{R}$ with five random jumps in $[0,0.5]$:

$$F(x) = \sum_{i=1}^{5} g_i \mathbf{1}_{(t_i,1]}(x) + \sin(8\pi x), \qquad t_i \overset{\text{iid}}{\sim} \text{Unif}(0,0.5), \quad g_i \overset{\text{iid}}{\sim} \text{Unif}(0,5) \tag{10}$$

and data according to

$$y_i = F(x_i) + N(0,0.25^2), \quad x_i \overset{\text{iid}}{\sim} \text{Unif}(0,1), \quad i=1,\dots,1000.$$

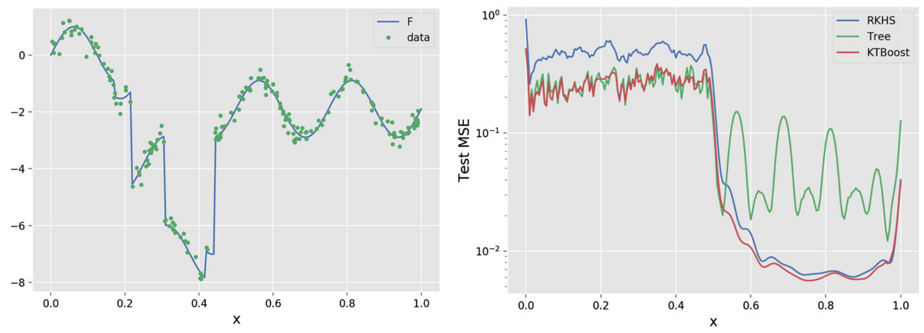

Fig. 2 An example of a random function with five random jumps in [0, 0.5] and corresponding observed data (left plot) and pointwise mean square error (MSE) for tree and kernel boosting as well as the combined KTBoost algorithm (right plot)

In Fig. 2 on the left-hand side, an example of such a function and corresponding data is shown. We simulate 1000 times such random functions as well as training, validation, and test data of size $n = 1000$. For each simulation run, learning is done on the training data. The number of boosting iterations is chosen on the validation data with the maximum number of boosting iterations being $M = 1000$. We use a learning rate of $\nu = 0.1$ as this is a reasonable default value [8] and trees of depth = 1 as there are no interactions. Further, for the RKHS ridge regression, we use a Gaussian kernel

$$K(x_1, x_2) = \exp\left(-\|x_1 - x_2\|^2/\rho^2\right), \tag{11}$$

with $\rho = 0.1$ and $\lambda = 1$. In Fig. 2 on the right-hand side, we show the pointwise test mean square error (MSE) for tree and kernel boosting as well as the combined KTBoost algorithm. We observe that tree boosting performs better than kernel boosting in the area where the discontinuities are located and, conversely, kernel boosting outperforms tree boosting on the smooth part. The figure also clearly shows that KTBoost outperforms both tree and kernel boosting as it achieves the MSE of tree boosting on the interval with jumps and the MSE of kernel boosting on the smooth part.

For the purpose of illustration, we have considered a one-dimensional example. However, in practice discontinuities, or strong non-linearities, as well as smooth parts are likely to occur at the interaction level in higher dimensions of a feature space.

4.2 Real-World Data

In the following, we compare the KTBoost algorithm with tree and kernel boosting using the following Delve, Keel, Kaggle, and UCI data sets: abalone, ailerons, bank8FM, elevators, energy, housing, liberty, NavalT, parkinsons, puma32h, sarcos, wine, adult, cancer, ijcnn, ionosphere, sonar, car, epileptic, glass, and satimage. Detailed information on the number of samples and features can be found in Table 1. We consider both regression as well as binary and multiclass classification data sets. Further, we include data sets of different sizes in order to investigate the performance on both smaller and larger data sets, as small- to moderately-sized data sets continue to be widely used in applied data science despite the recent focus on very large data sets in machine learning research. We use the squared loss for regression, the logistic regression loss for binary classification, and the cross-entropy loss with the softmax function for multiclass classification.

Table 1 Summary of data sets

Data	# classes	Nb. samples	Nb. features
abalone	Regression	4177	10
ailerons	Regression	13,750	40
bank8FM	Regression	8192	8
elevators	Regression	16,599	18
energy	Regression	768	8
housing	Regression	506	13
liberty	Regression	50,999	117
NavalT	Regression	11,934	16
parkinsons	Regression	5875	16
puma32h	Regression	8192	32
sarcos	Regression	48,933	21
wine	Regression	4898	11
adult	2	48,842	108
cancer	2	699	9
ijcnn	2	141,691	22
ionosphere	2	351	34
sonar	2	208	60
car	4	1728	21
epileptic	5	11,500	178
glass	7	214	9
satimage	6	6438	36

For the regression data sets, we use gradient boosting, and for the classification data sets, we use boosting with Newton updates since this can result in more accurate predictions [41]. For some classification data sets (adult, ijcnn, epileptic, and satimage), Newton boosting is computationally infeasible on a standard single CPU computer with the current implementation of KTBoost, despite the use of the Nyström method with a reasonable number of Nyström samples, say 1000, since the weighted kernel matrix in Eq. (9) needs to be factorized in every iteration. We thus also use gradient boosting for these data sets. Technically, it would be possible for these cases to learn the trees using Newton's method, or using the hybrid gradient-Newton boosting version of Friedman [18], but this would result in an unfair comparison that is biased in favor of the base learner which is learned with the better optimization method. For the larger data sets (liberty, sarcos, adult, ijcnn), we use the Nyström method described in Sect. 3.1. Specifically, we use $l = 1000$ Nyström samples, which are uniformly sampled from the training data. In general, the larger the number of Nyström samples, the lower the approximation error but the higher the computational costs. Williams and Seeger [43] reports good results with $l \approx 1000$ for several data sets. All calculations are done with the Python package `KTBoost` on a standard laptop with a 2.9 GHz quad-core processor and 16 GB of RAM.

All data sets are randomly split into three non-overlapping parts of equal size to obtain training, validation and test sets. Learning is done on the training data, tuning parameters are chosen on the validation data, and model comparison is done on the holdout test data. All input features are standardized using the training data to have approximately mean zero and variance one. In order to measure the generalization error and approximately quantify variability in it, we use ten different random splits of the data into training, validation and test sets. We note that when using a resampling approach, standard statistical tests, such as a paired t-test, cannot be used to do a pairwise comparison of the different algorithms on a dataset basis since training and test datasets in different splits are dependent due to overlap [3,13,14]. In particular, this can result in biased standard error estimates for the generalization error.

For the RKHS ridge regression, we use again a Gaussian kernel; see Eq. (11). Concerning tuning parameters, we select the number of boosting iterations M from $\{1, 2, \dots, 1000\}$, the learning rate ν from $\{1, 10^{-1}, 10^{-2}, 10^{-3}\}$, the maximal depth of the trees from $\{1, 5, 10\}$, and the kernel ridge regularization parameter λ from $\{1, 10\}$. Further, the kernel range parameter ρ is chosen using k-nearest neighbors distances as described in the following. We first calculate the average distance of all k-nearest neighbors in the training data, where k is a tuning parameter selected from $\{5, 50, 500, 5000, n-1\}$ and n is the size of the training data. We then choose ρ such that the kernel function has decayed to a value of 0.01 at this average k-nearest neighbors distance. This is motivated by the fact that for a corresponding Gaussian process with such a covariance function, the correlation has decayed to a level of 1% at this k-nearest neighbor distance. If the training data contains less than 5000 (or 500) samples, we use $n-1$ as the maximal number for the k-nearest neighbors. In addition, we include ρ which equals the average $(n-1)$-nearest neighbor distance. The latter choice is done in order to also include a range which results in a kernel that decays slowly over the entire space. For the large data sets where the Nyström method is used, we calculate the average k-nearest neighbors distance based on the Nyström samples. I.e., in this case, the maximal k equals $l-1$.

The results are shown in Table 2. For the regression data sets, we show the average test mean square error (MSE) over the different sample splits, and for the classification data sets, we calculate the average test error rate (=misclassification rate). The numbers in parentheses are approximate standard deviations over the different sample splits. In the last row, we report the average rank of every method over the different data sets. We find that KTBoost achieves higher predictive accuracy than both tree and kernel boosting for the large majority of data sets. Specifically, KTBoost has an average rank of 1.24 and achieves higher predictive accuracy than both tree and kernel boosting for seventeen out of twenty-one data sets. A Friedman test with an Iman and Davenport correction [25] gives a p value of 7.84×10^{-6} which shows that the differences in the three methods are highly significant. We next assess whether the pairwise differences in accuracy between the different methods are statistically significant using a sign test. Further, we apply a Holm–Bonferroni correction [21] to account for the fact that we do multiple tests. Despite the sign test having low power and the application of the conservative Holm–Bonferroni correction, KTBoost is highly significantly better than both tree and kernel boosting with adjusted p values below 0.01. The difference between kernel and tree boosting is not significant with both the adjusted and non-adjusted p values being above 0.1 (result not tabulated).

Note that we do not report the optimal tuning parameters since this is infeasible for all combinations of data sets and sample splits, and aggregate values are not meaningful since different tuning parameters often compensate each other in a non-linear way (e.g., number of iterations, learning rate, and tree depth or kernel regularization λ). Further, it is also difficult to concisely summarize the composition of the ensembles in terms of different base learners as a base learner that is added in an earlier boosting stage is more important than one that is added in a later stage [9], and the properties of the base learners also depend on the chosen tuning parameters. We also note that one can also consider additional tuning parameters. For trees, this includes the minimal number of samples per leaf, row and column sub-sampling,

Table 2 Comparison of KTBoost with tree and kernel boosting using test mean square error (regression) and test error rate (classification)

Data	KTBoost	Tree	Kernel
abalone	4.65 (0.248)	5.07 (0.261)	**4.64** (0.255)
ailerons	**2.64e−08** (6.19e−10)	8.11e−08 (2.39e−09)	2.64e−08 (6.19e−10)
bank8FM	**0.000915** (4.02e−05)	0.000945 (2.47e−05)	0.000945 (5.83e−05)
elevators	**4.83e−06** (2.9e−07)	5.66e−06 (1.44e−07)	5.18e−06 (3.89e−07)
energy	**0.282** (0.0372)	0.335 (0.093)	1.3 (0.377)
housing	**12.7** (3.19)	15.1 (3.23)	13.6 (2.51)
liberty	**14.5** (0.323)	14.5 (0.314)	15.2 (0.345)
NavalT	**6.51e−09** (1.15e−09)	1.15e−06 (1.58e−07)	6.51e−09 (1.15e−09)
parkinsons	**73.3** (1.98)	81.1 (2.44)	73.3 (1.91)
puma32h	**6.5e−05** (2.27e−06)	6.51e−05 (2.13e−06)	0.000695 (2.2e−05)
sarcos	**7.99** (0.206)	9.6 (0.207)	17.8 (0.586)
wine	**0.444** (0.012)	0.471 (0.0169)	0.506 (0.0106)
adult	0.128 (0.00295)	**0.128** (0.00313)	0.163 (0.00512)
cancer	0.0362 (0.00744)	0.0415 (0.0153)	**0.0358** (0.0107)
ijcnn	**0.0122** (0.000685)	0.0123 (0.000702)	0.0387 (0.00516)
ionosphere	**0.0872** (0.017)	0.103 (0.0226)	0.107 (0.0239)
sonar	0.194 (0.0394)	0.223 (0.05)	**0.193** (0.0491)
car	**0.0399** (0.00505)	0.0411 (0.00685)	0.041 (0.00624)
epileptic	**0.354** (0.00612)	0.373 (0.00614)	0.442 (0.0265)
glass	**0.308** (0.0711)	0.315 (0.0589)	0.344 (0.0581)
satimage	**0.089** (0.00452)	0.112 (0.00504)	0.0903 (0.00417)
Average rank	1.24	2.48	2.29
p val Friedman test	7.84e−06		
Adj. *p* val sign test		6.29e−05	0.00885

The smallest value are in boldface. In parentheses are approximate standard deviations. Below are average ranks of the methods over the different datasets. A p value of a Friedman test with an Iman and Davenport correction for comparing the different algorithms is also reported. The last row shows Holm–Bonferroni corrected p values of sign tests for pairwise comparison of the KTBoost algorithm with tree and kernel boosting

and penalization of leave values, and for the kernel regression, this includes the smoothness of the kernel function, or, in general, the class of kernel functions. One could also use different learning rates for the two types of base learners. Due to limits on computational costs, we have not considered all possible choices and combinations of tuning parameters. However, it is likely that a potential increase in predictive performance in either tree or kernel boosting will also result in an increase in accuracy of the combined KTBoost algorithm. We also note that in our experimental setup, the tuning parameter grid for the KTBoost algorithm is larger compared to the tree and kernel boosting cases. This seems inevitable in order to allow for the fairest possible comparison, though. Restricting one type of tuning parameters for the combined version but not for the single base learner case seems to be no alternative. Somewhat alleviating this concern is the fact that, in the above simulation study, we also find outperformance when not choosing tuning parameters using cross-validation, and on the downside, a larger tuning parameter grid might potentially also lead to overfitting. Finally, we remark that we have also considered to compare the risk of the un-damped base learners

$$R\left(F_{m-1} + f_m^T(x)\right) \leq R\left(F_{m-1} + f_m^K(x)\right)$$

in line 5 of Algorithm 1 when selecting the base learners that is added to the ensemble, and we obtain very similar results (see supplementary material).

5 Conclusions

We have introduced a novel boosting algorithm, which combines trees and RKHS functions as base learners. Intuitively, the idea is that discontinuous trees and continuous RKHS functions complement each other since trees are better suited for learning rougher parts of functions and RKHS regression functions can better learn smoother parts of functions. We have compared the predictive accuracy of the KTBoost algorithm with tree and kernel boosting and have found that KTBoost achieves significantly higher predictive accuracy compared to tree and kernel boosting.

Future research can be done in several directions. First, it would be interesting to investigate to which extent other base learners such as neural networks [23,31] are useful in addition to trees and kernel regression functions. Generalizing the KTBoost algorithm using reproducing kernel Kreĭn space (RKKS) learners [32,33] instead of RKHS learners can also be investigated. Further, theoretical results such as learning rates or bounds on the risk could help to shed further insights on why the combination of trees and kernel machines leads to increased predictive accuracy. Finally, it would be interesting to compare the KTBoost algorithm on very large data sets using different strategies for reducing the computational complexity of the RKHS part. Several potential strategies on how this can be done are briefly outlined in Sect. 3.1.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s11063-021-10434-9.

Acknowledgements We would like to thank Christoph Hirnschall for helpful feedback on the article. This research was partially supported by the Swiss Innovation Agency-Innosuisse (25746.1 PFES-ES).

Funding Open Access funding provided by Hochschule Luzern

References

1. Belkin M, Hsu DJ, Mitra P (2018a) Overfitting or perfect fitting? Risk bounds for classification and regression rules that interpolate. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R (eds) Advances in neural information processing systems, vol 31. pp 2306–2317

2. Belkin M, Ma S, Mandal S (2018b) To understand deep learning we need to understand kernel learning. In: Dy J, Krause A (eds) Proceedings of the 35th international conference on machine learning, volume 80 of proceedings of machine learning research. pp 541–549
3. Bengio Y, Grandvalet Y (2004) No unbiased estimator of the variance of k-fold cross-validation. J Mach Learn Res 5:1089–1105
4. Berlinet A, Thomas-Agnan C (2011) Reproducing kernel Hilbert spaces in probability and statistics. Springer, Berlin
5. Bevilacqua M, Faouzi T, Furrer R, Porcu E et al (2019) Estimation and prediction using generalized Wendland covariance functions under fixed domain asymptotics. Ann Stat 47(2):828–856
6. Blanchard G, Krämer N (2010) Optimal learning rates for kernel conjugate gradient regression. In:Advances in neural information processing systems. pp 226–234
7. Breiman L, Friedman J, Stone CJ, Olshen RA (1984) Classification and regression trees. CRC Press, Boca Raton
8. Bühlmann P, Hothorn T (2007) Boosting algorithms: Regularization, prediction and model fitting. Stat Sci 22:477–505
9. Bühlmann P, Yu B (2003) Boosting with the l 2 loss: regression and classification. J Am Stat Ass 98(462):324–339
10. Cesa-Bianchi N, Conconi A, Gentile C (2004) On the generalization ability of on-line learning algorithms. IEEE Trans Inf Theory 50(9):2050–2057
11. Chen T, Guestrin C (2016) Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, pp 785–794
12. Dai B, Xie B, He N, Liang Y, Raj A, Balcan M-FF, Song L (2014) Scalable kernel methods via doubly stochastic gradients. In: Advances in neural information processing systems. pp 3041–3049
13. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
14. Dietterich TG (1998) Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput 10(7):1895–1923
15. Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: ICML, vol 96. Bari, Italy, pp 148–156
16. Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. J Comput Syst Sci 55(1):119–139
17. Friedman J, Hastie T, Tibshirani R et al (2000) Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). Ann Stat 28(2):337–407
18. Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Ann Stat 29:1189–1232
19. Gneiting T (2002) Compactly supported correlation functions. J Multivar Anal 83(2):493–508
20. Hayakawa S, Suzuki T (2020) On the minimax optimality and superiority of deep neural network learning over sparse parameter spaces. Neural Netw 123:343–361
21. Holm S (1979) A simple sequentially rejective multiple test procedure. Scand J Stat 6:65–70
22. Hothorn T, Bühlmann P, Kneib T, Schmid M, Hofner B (2010) Model-based boosting 2.0. J Mach Learn Res 11:2109–2113
23. Huang F, Ash J, Langford J, Schapire R (2018) Learning deep resnet blocks sequentially using boosting theory. ICML 80:2058–2067
24. Imaizumi M, Fukumizu K (2019) Deep neural networks learn non-smooth functions effectively. In:The 22nd international conference on artificial intelligence and statistics. pp 869–878
25. Iman RL, Davenport JM (1980) Approximations of the critical region of the fbietkan statistic. Commun Stat Theory Methods 9(6):571–595
26. Ke G, Meng Q, Finley T, Wang T, Chen W, Ma W, Ye Q, Liu T-Y (2017) Lightgbm: a highly efficient gradient boosting decision tree. In: Advances in neural information processing systems. pp 3149–3157
27. Ma S, Belkin M (2017) Diving into the shallows: a computational perspective on large-scale shallow learning. In: Advances in neural information processing systems. pp 3778–3787
28. Mason L, Baxter J, Bartlett PL, Frean MR (2000) Boosting algorithms as gradient descent. In: Advances in neural information processing systems. pp 512–518
29. Mendes-Moreira J, Soares C, Jorge AM, Sousa JFD (2012) Ensemble approaches for regression: a survey. ACM Comput Surv (CSUR) 45(1):10
30. Murphy KP (2012) Machine learning: a probabilistic perspective. The MIT Press. ISBN 0262018020, 9780262018029
31. Nitanda A, Suzuki T (2018) Functional gradient boosting based on residual network perception. ICML 80:3819–3828
32. Oglic D, Gaertner T (2018) Learning in reproducing kernel kreĭ spaces. In: International conference on machine learning. pp 3859–3867

33. Ong CS, Mary X, Canu S, Smola AJ (2004) Learning with non-positive kernels. In: Proceedings of the twenty-first international conference on machine learning. pp 81
34. Peng J, Aved AJ, Seetharaman G, Palaniappan K (2018) Multiview boosting with information propagation for classification. IEEE transactions on neural networks and learning systems 29(3):657–669
35. Ponomareva N, Radpour S, Hendry G, Haykal S, Colthurst T, Mitrichev P, Grushetsky A (2017) Tf boosted trees: a scalable tensorflow based framework for gradient boosting. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, pp 423–427
36. Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulin A (2018) Catboost: unbiased boosting with categorical features. In: Advances in neural information processing systems vol 31. Curran Associates, Inc, pp 6638–6648
37. Rahimi A, Recht B (2008) Random features for large-scale kernel machines. In: Advances in neural information processing systems. pp 1177–1184
38. Raskutti G, Wainwright MJ, Yu B (2014) Early stopping and non-parametric regression: an optimal data-dependent stopping rule. J Mach Learn Res 15(1):335–366
39. Schölkopf B, Smola AJ (2001) Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT Press, Cambridge
40. Schölkopf B, Herbrich R, Smola AJ (2001) A generalized representer theorem. In: International conference on computational learning theory. Springer, pp 416–426
41. Sigrist F (2021) Gradient and newton boosting for classification and regression. Expert Syst Appl (in press)
42. Sigrist F, Hirnschall C (2019) Grabit: Gradient tree-boosted tobit models for default prediction. J Bank Finance 102:177–192
43. Williams CK, Seeger M (2001) Using the Nyström method to speed up kernel machines. In: Advances in neural information processing systems. pp 682–688
44. Wyner AJ, Olson M, Bleich J, Mease D (2017) Explaining the success of adaboost and random forests as interpolating classifiers. J Mach Learn Res 18(48):1–33
45. Yao Y, Rosasco L, Caponnetto A (2007) On early stopping in gradient descent learning. Constr Approx 26(2):289–315
46. Zhang C, Bengio S, Hardt M, Recht B, Vinyals O (2017) Understanding deep learning requires rethinking generalization. In: International conference on learning representations
47. Zhang Y, Duchi J, Wainwright M (2013) Divide and conquer kernel ridge regression. In: Conference on learning theory. pp 592–617
48. Zhang Y, Duchi J, Wainwright M (2015) Divide and conquer kernel ridge regression: a distributed algorithm with minimax optimal rates. J Mach Learn Res 16(1):3299–3340